Q76054AP.BCR
OCTOBER 20, 2003

SYSTEM AND METHOD FOR SHORTENING COMPILING TIME OF BYTE CODES IN JAVA PROGRAM

This application claims the priority of Korean Patent Application No. 10-2003-0011366 filed on February 24, 2003, in the Korean Intellectual Property Office, the disclosure of which is incorporated herein in its entirety by reference.

BACKGROUND OF THE INVENTION

1. Field of Invention

[01] The present invention relates to a system and method for shortening the compiling time of byte codes in a Java program, and more particularly, to a system and method for shortening the compiling time of byte codes in a Java program, wherein byte codes frequently used upon execution of the Java program are compiled and resultant native codes are retrieved and executed upon future execution of the Java program, thereby shortening the compiling time of the native codes.

2. Description of the Related Art

[02] Generally, byte codes generated through a compiling process of Java program source codes are executed by a Java virtual machine (JVM). Recently, in order to enhance the execution speed of the Java virtual machine, methods in which a CPU interprets byte codes to directly executable native codes have been widely used.

[03] Among such methods, in the dynamic adaptive compilation (DAC), byte codes are interpreted and those that are categorized as bottleneck byte codes, which are frequently used byte codes, are compiled to native codes to be executed. That is, the DAC method compiles only frequently used byte codes to native codes, rather than all the byte codes in a program.

[04] However, there are problems in that the native codes generated during the execution of the byte codes are collected by a garbage collector, in case of lack of memory, and cannot be reused for future execution since all native codes are deleted after the Java program has been completely executed.

[05] Since most of the byte codes that have once been compiled to native codes tend to be repeatedly compiled for future execution, there is a problem in that the deletion of the generated native codes results in the repeated generation of the same native codes to be used in future executions. Thus, the conventional methods have problems in that in a system such as a mobile terminal with a low-performance CPU and a low-capacity battery, system response time is increased and the battery is rapidly exhausted.

SUMMARY

[06] The present invention is conceived to solve the aforementioned problems. An exemplary object of the present invention is to provide a system and method for shortening the compiling time of byte codes in a Java program, wherein relevant byte codes frequently used upon execution of the Java program are compiled and resultant native codes are stored so that the native

codes for the relevant byte codes are retrieved and executed upon future execution of the Java program.

[07] Another exemplary object of the present invention is to provide a system and method for shortening the compiling time of byte codes in a Java program, by which the compiling time and the execution time of the Java program are shortened in equipment with a low-performance CPU and a low-capacity battery, thereby reducing response time to a user and conserving battery power.

[08] According to an exemplary aspect of the present invention for achieving these objects, there is provided a system for shortening the compiling time of byte codes in a Java program, comprising a class loader unit for loading byte codes generated by compiling Java program source codes; a first memory unit for maintaining the byte codes loaded by the class loader unit and native codes generated by compiling the byte codes in an accessible state; a second memory unit for storing the native codes that are loaded into the first memory unit in the accessible state; a native code manager unit for searching the native codes stored in the second memory unit and loading the searched native codes into the first memory unit according to a request by a class loader unit; and an execution unit for executing the native codes that are loaded into the first memory unit in the accessible state.

[09] In addition, according to another exemplary aspect of the present invention, there is provided a method for shortening the compiling time of byte codes in a Java program, comprising the steps of loading compiled byte

codes by a class loader unit; requesting a native code manager unit to search native codes corresponding to the loaded byte codes; searching the requested native codes in a second memory unit; transmitting the searched native codes to a first memory unit; and executing the transmitted native codes by a code execution unit.

BRIEF DESCRIPTION OF THE DRAWINGS

[10] The above and other objects and features of the present invention will become apparent from the following description of illustrative, non-limiting embodiments given in conjunction with the accompanying drawings, in which:

[11] FIG. 1 is a block diagram schematically showing an exemplary system for shortening the compiling time of byte codes in a Java program; and

[12] FIG. 2 is a flowchart schematically illustrating an exemplary method for shortening the compiling time of byte codes in a Java program.

DESCRIPTION

[13] Hereinafter, illustrative, non-limiting embodiments of the present invention will be described in detail with reference to the accompanying drawings.

[14] FIG. 1 is a block diagram schematically showing an exemplary system for shortening the compiling time of byte codes in a Java program. The system comprises a class loader unit 100, a first memory unit 200, a native code manager unit 300, a second memory unit 400, an execution unit 500 and a garbage collector unit 600.

[15] The class loader unit 100 loads the byte codes generated by compiling Java program source codes. Here, the byte codes originally located in an auxiliary memory unit are loaded into a Java virtual machine.

[16] The first memory unit 200 maintains the byte codes loaded by the class loader unit 100 and the native codes generated by compiling the byte codes in an accessible state. That is, the byte codes loaded by the class loader unit 100 and the native codes generated by compiling the byte codes are stored in predetermined memory areas so that the execution unit 500 to be described later can access the stored native codes.

[17] The second memory unit 400 stores the native codes that are loaded into the first memory unit 200 in the accessible state.

[18] The native code manager unit 300 stores the native codes, which have been loaded into the first memory unit 200, in the second memory unit 400. The native code manager unit 300 searches the native codes stored in the second memory unit 400 according to a request by the class loader unit 100, and loads the searched native codes into the first memory unit 200. The native code manager unit 300 manages the native codes stored in the second memory unit 400 by using the least recently used (LRU) method. In the LRU method, rarely used native codes among the stored native codes are checked and then deleted in order of rarity of use thereof. Here, the rarely used native codes are deleted, for example, based on memory size and time when they were stored.

[19] The execution unit 500 executes the native codes and byte codes loaded into the first memory unit 200 in the accessible state, and comprises a

byte code interpreter 510, a runtime profiler 520, a native code compiler 530 and a native code executor 540.

[20] The byte code interpreter 510 interprets the byte codes loaded into the first memory unit 200 in the accessible state and executes the interpreted byte codes.

[21] The runtime profiler 520 checks whether the byte codes interpreted by the byte code interpreter 510 are frequently used byte codes and reports the check results to the native code compiler 530. The native code compiler 530 generates native codes by compiling the byte codes being interpreted. Further, the native code compiler 530 loads the compiled native codes into the first memory unit 200 if the runtime profiler 520 confirms that the byte codes are frequently used byte codes.

[22] The native code executor 540 executes the native codes loaded into the first memory unit by the native code manager unit 300.

[23] The garbage collector unit 600 automatically collects space occupied by unused codes in the first memory unit 200 so as to secure more space in the first memory unit 200. If a space shortage occurs in the first memory unit 200 even after the garbage collector unit 600 has collected the space occupied by the unused codes, the garbage collector unit 600 requests the native code manager unit 300 to store the native codes in the second memory unit 400.

[24] FIG. 2 is a flowchart schematically illustrating an exemplary method for shortening the compiling time of byte codes in a Java program according to the present invention.

[25] First, the class loader unit 100 loads the byte codes generated by compiling Java program source codes (S100) and requests the native code manager unit 300 to search for native codes corresponding to the loaded byte codes (S110). Then, the native code manager unit 300 searches for the requested native codes in the second memory unit 400 (S120).

[26] When the corresponding native codes are found in the second memory unit 400 (S130), the searched native codes are transmitted to the first memory unit 200 (S132). Then, the native code executor 540 executes the native codes transmitted to the first memory unit 200 (S134). Here, the native codes stored in the second memory unit 400 are native codes generated by compiling the frequently used byte codes in the Java program.

[27] In the present invention, the native codes stored in the second memory unit 400 are simply loaded into and executed in the first memory unit. Thus, there is an advantage in that it is not necessary to interpret the frequently used byte codes every time, thereby eliminating the byte code interpretation process.

[28] Meanwhile, if there are no native codes corresponding to the relevant byte codes as a result of the search of the second memory unit 400 by the native code manager unit 300, the class loader unit 100 transmits the loaded byte codes to the first memory unit 200 (S140).

[29] When the byte codes are loaded into the first memory unit 200, the byte code interpreter 510 interprets the loaded byte codes to be executed (S150).

[30] While the byte code interpreter 510 interprets the byte codes, the runtime profiler 520 checks whether the byte codes being interpreted are the frequently used byte codes (S160). The check results are then transmitted to the native code compiler 530.

[31] If it is determined that the byte codes are frequently used byte codes as a result of the checking process (S170), the interpreted byte codes are transmitted to the native code compiler 530 and compiled to generate corresponding native codes (S172). Here, the native codes are transmitted to the first memory unit 200 and then stored in the second memory unit 400 by the native code manager unit 300 (S174). At this time, the native codes that are stored in the second memory unit 400 are managed by the native code manager unit 300 according to the LRU method. That is, since there are limitations in storage areas of the second memory unit 400, the LRU method is employed to manage the stored native codes.

[32] However, byte codes that are not frequently used or are defined as having been previously unused, will be interpreted by the byte code interpreter 510 to be executed (S180).

[33] The garbage collector unit 600 automatically collects the space occupied by the unused codes so as to secure more space in the first memory unit 200. In addition, if the first memory unit 200 lacks space even after the unused codes are processed, the native code manager unit 300 is requested to store the native codes, which have been loaded into the first memory unit 200,

in the second memory unit 400, thereby securing more space in the first memory unit 200.

[34] According to the present invention, relevant byte codes frequently used upon execution of a Java program are compiled and resultant native codes are stored so that the native codes for the relevant byte codes are retrieved and executed upon future execution of the Java program. Thus, there is an advantage in that the time required for generating the native codes by compiling the byte codes can be shortened.

[35] Further, there are advantages in that the compiling time and the execution time of the Java program are shortened in equipment such as a cellular phone with a low-performance CPU and a low-capacity battery, thereby reducing response time to a user and conserving battery power.

[36] The present invention has been described in connection with the illustrative, non-limiting embodiments thereof shown in the accompanying drawings and are mere examples of the present invention. It can also be understood by those skilled in the art that various changes and modifications thereof can be made thereto without departing from the scope and spirit of the present invention defined by the claims. Therefore, simple changes to the embodiments of the present invention fall within the scope of the present invention.